Abstract

We present new results achieved in the application of incremental graph composition algorithm, in particular using the label pushing method to further reduce the final graph size. In our previous work we have shown that the incremental composition is an efficient alternative to the conventional finite state transducer (FST) determination-composition-minimization approach, with some limitations. One of the limitations was that the word labels must stay aligned with the actual word ends. We describe an updated version of the algorithm which allows us to push the word labels relatively to the word ends to increase the effect of the minimization. The size of resulting graph is now very close to the ones produced by the conventional FST approach with label pushing.

Index Terms: speech recognition, static graph composition

1. Introduction

Weighted finite state transducers are commonly used in state-of-the-art speech recognition systems. As originally proposed by Mohri at al [2], they provide a solid theoretical framework for the operations needed for decoding graph construction. A decoding graph is the result of a composition

\[ S = HC \circ L \circ G, \]

where \( G \) represents a language model, \( L \) represents a pronunciation dictionary and \( HC \) translates phone sequences to context dependent HMM states. After minimization, such a graph leads to the most efficient decoder implementations. Two main issues arise though; the complexity of the composition and the size of the final graph. The complexity of the composition comes mainly from the need to model co-articulation. A decision tree is typically used to assign a GMM to each HMM state given its surrounding phonetic context. The complexity of graph construction grows significantly with the size of the modeled context, i.e., the number of phones the tree can see on both sides of the modeled phone. To reduce the complexity of graph construction, the context is often limited to tri-phones, left cross-word or even word internal. Direct application of the composition steps can be very memory intensive and determination and minimization may be needed after each step.

We have shown previously that the decoding graph can be efficiently built using incremental composition [1]. The main advantage of this technique is that by combining the composition and minimization steps together, creation of possibly large intermediate graphs is avoided. For acoustic models with word-internal or left-cross word context modeling incrementally composed graphs are minimal. An interesting property of this algorithm is that the minimization is achieved by application of a faster acoustic minimization even in cases where \( G \) is cyclic (e.g. for n-gram language models).

In this paper we present an updated version of the incremental composition algorithm, with the output label pushing [?] incorporated into the graph construction. Because the available space does not allow us the present the original algorithm in full detail, we encourage the reader to get familiar with our previous work [1]. In the rest of this paper, first we show the differences between the traditional and incremental composition. Then we describe the changes we have made to the algorithm to enable label pushing, followed by a discussion of the effectiveness of label pushing in various situations.

2. Label pushing effect on the composition

A weighted finite state transducer (WFSA) \( A = (\Sigma, Q, E, i, F, \lambda, \rho) \) over a semiring \( K = (\mathbb{K}, \oplus, \otimes, 0, 1) \) is given by an alphabet input \( \Sigma \), output alphabet \( \Omega \), a finite set of states \( Q \), a finite set of transitions \( E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q \), where \( i \in Q \) is a set of initial states, and \( F \subseteq Q \) is a set of final states. Let \( \lambda : \Sigma \rightarrow \mathbb{K} \) be the initial weight and \( \rho : Q \rightarrow \mathbb{K} \) be a final weight function.

A composition of two weighted transducers \( A \) and \( B \) is defined by

\[ (A \circ B)(u, v) = \bigoplus_{w \in \Omega^*} A(u, s) \otimes B(s, v), \]

where \( A(u, v) \) is a mapping from input-output pairs of strings from alphabets \( \Sigma \) and \( \Omega \) to weights, \( T : \Sigma^* \times \Omega^* \rightarrow K \) as defined by the transducer \( A \).

The size of the composed transducer can be usually reduced by a minimization algorithm, which finds and merges sets of equivalent states. Before the minimization algorithm is applied, the weights and the output labels can be pushed to increase the number of equivalent states in the graph. The weight pushing essentially normalizes arc weights leaving each state (depending on the semiring). The label pushing moves the output labels towards the initial states, which allows for common word suffixes to be merged together. We will illustrate the composition step on an example of \( S = L \circ G \). Figure 1 shows an example of a word grammar (shown as an acceptor with input labels only) to be composed with a lexicon transducer mapping phone sequences to words in Figure 2. Both transducers are minimal, it can be seen that because of the label pushing, the common suffix of words nine and seven has been merged. We have introduced a special symbol \( \bot \) to mark the original word ends. It is required by our targeted decoder [3] to determine the correct word alignments. Introduction of these special labels presents a limitation of our method in comparison to the standard composition, since the output labels cannot be pushed outside of the word boundaries and the effect of minimization can be reduced in some rare cases.

The states of the composed transducer created by the standard composition algorithm correspond to pairs of states \((p, q) \in Q_L \times Q_G \) and an arc is created for each case when the output label of an arc from \( p \) matches the input label of an arc from \( q \). To apply this algorithm for our specific \( L \) and \( G \), we need to formally augment \( G \) with arcs accepting \( \epsilon \) and \( \bot \) labels as shown in Figure 3. Result of the composition is show in Figure 4a. It is sequential (i.e. deterministic in the input labels) if \( G \) and \( L \) are sequential but not minimal. The result of the minimization is shown in Figure 4b.
Starting with an empty graph $S$, result of each local composition $S_k$ is merged with $S$ while the new $S$ is immediately minimized. The minimization is performed by testing each state of $S_k$ for equivalence with states of $S$. If no equivalent state is found, the state is added into $S$. All locally composed states are processed in the post-order fashion which guarantees that all children of any state being tested for equivalence already exist in $S$. When all states of $Q_G$ are locally composed with $L$, the resulting final graph is sequential and minimal.

Figure 5 shows results of the composition application to each state of $G$. After the first step (a), the set of arcs leaving state 1 is replaced by a subset of $L$ representing all pronunciations generated by these arcs. In the next step (b) state 2 is processed. When state 3 is processed in the final step (c), many of the new states already have their equivalents in the final graph, so only arcs and states shown as thick lines need to be added. It can be seen that in contrast with the traditional composition algorithm, no sets of equivalent states are created.

3. Incremental composition

We will first explain the incremental composition concept for the case of $S = L \circ G$ (which is directly applicable to models with no cross-word context). The basic idea of the incremental approach is to perform the composition locally for each state $q_k \in Q_G$ of the grammar $G$ creating graph segments

$$S_k = ((HC \circ L) \circ G_k),$$

where $G_k$ is a transducer formed by arcs leaving the state $q_k$ only.
from their original position, they are replaced by the word end symbol $\perp$, in all subsequent moves they are replaced by the $\epsilon$ symbol. Application of the subtree selection is illustrated in Figure 6. Figure 6(a) shows $L$ as a tree with leaves for words (\text{four, five, seven}) with weights (1.00, 0.50, 0.23) marked for selections (octagons). The resulting subtree is shown in Figure 6(b), where all selected states of the selected words are marked as octagons and arcs as solid lines.

To address the cross-word context, the composition is done in two passes. In the first pass the incremental composition is applied using the subtree selection algorithm to all states of $G$ building one prefix tree $L$ for each unique left context $c_L$ with multiple leaves representing unique right context variants of each pronunciation (in our examples we assume each word has one pronunciation).

The subtree selection algorithm is applied in the same way, with the exception of the leaf states (word ends) and the root states. These states are left unconnected, using the original states of $G$ as their temporary state ids. The missing connections will be finalized in the second pass. We need to collect the following information during the first pass:

1. a map $\rho: (q, c_L) \mapsto q_{c_L}$ which translates a pair of the left context identifier $c_L$ and $q \in Q_G$ to the unconnected root of each inserted subtree,
2. a set of the right contexts for each leaf of the subtree.

We use a hash table for the left context map and an encoding scheme described below to augment the word end symbol $\perp$ with the right context set information.

Let us define a mapping $\lambda_{c_L}: \Sigma_G \mapsto P^{L_{c_L}}$ where $P$ is the phone set and $L_{c_L}$ is the size of the left context. This mapping maps the pronunciation to its last $L_{c_L}$ phones. For example $\lambda_{\text{six}}(K, S) = 2$, $L_{\text{six}} = 2$. Similarly we define $\lambda_{c_R}: \Sigma_G \mapsto P^{L_{c_R}}$ which maps the pronunciation to its first $L_{c_R}$ phones.

Let us define $C_S \subset P(P^{L_{c_S}})$ where $P(A)$ denotes power set of $A$ and $L_{c_S}$ is the size of the right context. For example $\{(\text{F, AY}), (\text{S, HI})\} \in C_S$ for $L_{\text{ay}} = 2$. In the rest of the paper we will assume $L_{c_L} = L_{c_R} = 1$, though the algorithm is applicable to larger context sizes at the price of higher computational cost.

We can now construct a left-context-specific state prefix tree for each context $c_L \in \{\lambda_{c_L}(u): u \in \Sigma_G\}$. During the tree build, for each pronunciation we need to consider a state sequence for each possible right context $c_R \in \{\lambda_{c_R}(u): u \in \Sigma_G\}$. Since some of the state sequences will be equivalent, each leaf of the tree will represent a set of specific right contexts $Z_{c_S} \in C_S$. Though cardinality of $C_S$ is $2^{P^{L_{c_S}}}$, the actual size of the observed subset was below 200 even for a large vocabulary. We can assign an index $p_{c_S}$ to each unique $Z_{c_S}$ using a hash table. This index can then be encoded into the label of the leaf leading into leaf together with the pronunciation index. The resulting tree is shown in Figure 7. The root state of the tree carries the information about the left context in which the tree was built, and each leaf state arc provides information about which right context it can be used in.

Each left-context-specific instance of the tree is applied only to those states which are affected by this context. All leaf states (right-context variants) corresponding to an active pronunciation are selected. This may lead to the creation of unreachables, but this is a concern for grammar based GS only. In back-off n-gram models, due to the back-off arcs, all right-context variants will be used eventually. When leaf states are inserted into the graph at the beginning of sub-tree selection, the original destination state indices are used as in the word-internal context case, because their context-dependent instances may not exist yet. The correct destination will be determined in the second pass (with the exception of $\epsilon$-arcs). When the root state is reached at the end of sub-tree selection, its index will represent the left-context-dependent state index. This index is inserted into a mapping table $p: (q, c_L) \mapsto q_{c_L}$ to be used in the second pass.

The main difference introduced by the label pushing is that the leaf labels no longer carry the pronunciation identity, which will be needed in the second pass to determine which left context it represents. We solve this by replacing the pronunciation labels with a left-context variant of the word end $\perp$ symbol. This is combined with the right context information, so a single label is used to encode the information about the placement of the word end, its left context instance and the right context set it can connect to.

Label pushing is not applied all the way to the root of the tree, but it is stopped one level before reaching the root. The reason for that is that we will need to know the phone identity of all arcs at the root state during the second pass.

A special treatment needs to be applied to $\epsilon$-arcs in $G$. They are used in $n$-gram models to represent back-off transitions and should not be confused with output symbols in $L$. If they are considered as part of the pronunciation vocabulary, $G$ is still deterministic. They don’t create a context on their own; they only transfer it from one root state to another.

Figure 9: Segment of $G$

The connections created by the second pass is shown in Figure 8, representing segment of the final graph corresponding to a segment of $G$ in Figure 9. All arcs and states in solid black were created in the first pass. The unconnected states represent the roots of inserted left-context instances of the prefix tree for states 2 and 3. Arcs leaving those states (in dotted black) are not included in the final graph, they were used only temporarily by the first pass. As thicker green arcs are shown the arcs created in the second pass by selecting only those arcs leaving the left-context instances prefix tree root satisfying the right context constraints of arcs connecting to the this root. The details of the second pass connection algorithm can be again found in [1].

In the final step, the graph needs to be globally minimized. Due to the right cross-word constraints, connections created during the second pass may create states which could not have been tested for equivalence during the first pass (this situation does not occur when only word-internal or left cross-word constraints are used). Our experiments showed that the result of the second pass is usually close to being minimal. As part of the final minimization step implementation, we have experimented with several implementations of the global weight pushing step. We have found very little difference in the deciding speed when either tropical or log semiring was used for a typical $n$-gram model (but for certain types of rule based grammars the difference can be significant). We have found the following implementation based on Dijkstra’s algorithm applicable in tropical semiring to be very efficient. It uses a heap of tuples $(q, C(q))$, where $C(q)$ is a cost of reaching the final state from the state $q$ over some path, initialized with the final state and its cost. The heap keeps tuples sorted by $C(q)$. When it is finished it fills the array $p_{cost}$ with the cost of reaching the final state for each $q \in Q_S$. The function cost($p, q$) returns the lowest cost of transition from state $p$ to state $q$.

Algorithm 1

1. $heap \leftarrow \{(q, cost(q)): p_{cost}[q] \leftarrow \infty, q \in Q_S\}$
2. while heap.empty()
Some states can be inserted into the heap multiple times, but all such instances but the one with lowest cost are discarded. The size of the heap is significantly smaller than the size of the graph and the cost of the algorithm is close to $O(|Q_s|)$.

### 4. Results

Table 1 shows the resulting size of the graph in terms of the number of states and arcs for several acoustic and language models. In our graph encoding scheme, the number of arcs is the main determining factor for the actual graph file size. The English model has 90k pronunciations, the Arabic one has 2.6M (the same models and 4-gram LMs were used to report results in [1]). All models use one GMM for each state and context (the same models and 4-gram LMs were used to report results in [1]).

Table 1: Comparison of the graph size (in MB) with and without label pushing

<table>
<thead>
<tr>
<th># of AM states</th>
<th>English</th>
<th>vowel</th>
<th>Arabic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2196</td>
<td>30.0M</td>
<td>44.7M</td>
<td>448k</td>
</tr>
<tr>
<td>6000</td>
<td>44.7M</td>
<td>60.0M</td>
<td>118.5M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM size</th>
<th>with pushing</th>
<th>w/o pushing</th>
</tr>
</thead>
<tbody>
<tr>
<td>states/arcs</td>
<td>448k</td>
<td>1.6M</td>
</tr>
<tr>
<td></td>
<td>848k</td>
<td>2.7M</td>
</tr>
<tr>
<td>1-gram size</td>
<td>4.2M</td>
<td>4.8M</td>
</tr>
<tr>
<td></td>
<td>28.8M</td>
<td>25.3M</td>
</tr>
<tr>
<td></td>
<td>6.0M</td>
<td>9.5M</td>
</tr>
<tr>
<td></td>
<td>30.0M</td>
<td>50.5M</td>
</tr>
<tr>
<td></td>
<td>48.4M</td>
<td>76.5M</td>
</tr>
</tbody>
</table>

![Figure 7: Context dependent state prefix tree](image)

![Figure 8: Cross word connections](image)

![Figure 10: Decoding speed comparison](image)

### 5. References


